Integrace DSpacu do IS ČVUT – technologický pohled

Vypracoval: Marek Moos (VIC ČVUT)

marek.moos@vc.cvut.cz

Datum: 24. 3. 2009

Verze: 1.0

1 Úvod

V tomto textu se zamýšlíme výhradně nad technologickými požadavky a možnostmi integrace DSpacu do informačního systému ČVUT a popisujeme realizované řešení. Hlavní pozornost věnujeme SOA (Service Oriented Architecture) přístupu. Jsme si vědomi, že výraz SOA se často používá značně vágně a bývá někdy považován za „buzzword“. Proto všechny SOA principy, které zde zmiňujeme, uvádíme na konkrétních příkladech integrace DSpacu. Pomáhají nám zde zvládnout požadavky, které přirozeně vyplývají z povahy úlohy. Tyto požadavky podrobně rozebíráme. Na závěr se zamýšlíme, kde má stávající řešení mezery a jak jej vylepšit.

2 Co chceme udělat

Cílem je integrovat DSpace s některými komponentami informačního systému ČVUT. Konkrétně chceme:

2.1 Automaticky synchronizovat strukturu univerzity (fakulty, katedry…) vedenou ve Výměníku (komponenta v IS ČVUT) se strukturou v DSpacu.

2.2 Automaticky přenášet studentské práce (bakalářky, diplomky, disertace), včetně jejich metadat, z KOSu do DSpacu.

Dále pak chceme poskytnout natolik obecné řešení, které umožní potenciálně jiným klientům DSpacu, než je KOS a Výměník, využívat služby pro updatování struktury DSpacu a importování prací (itemů) do DSpacu.

3 Jaké z toho plynou požadavky

3.1 Technologické požadavky

3.1.1 Je třeba počítat s tím, že některé části klientů DSpacu a DSpacu samotného mohou být ve vzájemně fyzicky izolovaných prostředích (firewally, demilitarizované zóny …). Potom např.:

- Klient nemá přístup do databáze DSpacu a naopak
- Klient nemá přístup na file system DSpacu a naopak

3.1.2 Klient a DSpace mohou být implementovány na různých platformách (např. klient - .NET, DSapce - Java). Pro komunikaci je proto třeba zvolit na platformě nezávislý protokol.

3.1.3 Velikost Importovaných itemů může být potenciálně značná a import může trvat dlouho. Navíc komunikační kanál mezi klientem a DSpacem nemusí vždy fungovat. Uvažujme například situaci, kdy použijeme pro transport jednu http konekci, kterou odešleme item a po jeho zpracování vrátíme v té samé konekci odpověď. Pokud doba zpracování itemu trvá déle než max. doba konekce, klientovi se vrátí chyba – „vypršela konekce“. Přesto může být item uložen v DSpacu korektně. Tímto by vznikl inkonzistentní stav.

Proto požadujeme možnost importovat itemy asynchronně. Ještě lépe by bylo mít možnost způsob transportu konfigurovat.

3.1.4 Představme si například, že chceme vytvořit proces “složení státní závěrečné

zkoušky", jehož součástí bude export diplomky do DSpacu. Z pohledu bezpečnostního chceme, aby se spuštěný proces nejprve autentizoval (ověření identity) a následně, aby se v jednotlivých krocích autorizoval (ověření přístupových práv). Vidíme, že import itemu do DSpacu je zde v určitém bezpečnostním kontextu celého procesu. Z pohledu transakčního chceme, aby všechny části procesu proběhly v jedné distribuované transakci. Import itemu do DSpacu je zde v určitém transakčním kontextu.

Služby DSpacu tedy mohou být využity v různých kontextech (např. v různých bezpečnostních, transakčních …). Požadujeme zde, aby příslušné chování (policy) v kontextu (security policy, transaction policy…) bylo konfigurovatelné a nezávislé na implementaci služby.

3.2 Metodologické požadavky

V úvodu tohoto dokumentu si klademe za cíl poskytnout služby DSpacu v obecnějším tvaru, aby byly potenciálně využitelné i pro jiné procesy. Z toho vyplývají následující požadavky:

3.2.1 Chceme jasnou a přesnou definici (kontrakt), co daná služba poskytuje (z toho potom také plyne, co neposkytuje). Dále chceme, aby tato definice byla vyjádřena formou, která je obecně srozumitelná a zpracovatelná. Jde nám o to, aby rozhodnutí, zda a jak sužbu použít, šlo učinit na základě příslušného kontraktu (netřeba hrabat se v kódu určité implementace). Dále můžeme požadovat, aby šlo daný kontrakt a jeho implementace vyhledat v nějakém repository.

3.2.2 Abychom docílili obecnosti a tedy i znuvupoužitelnosti služby, požadujeme, aby byla služba co možná agnostická vůči kontextu, ve kterém se vyskytuje. Neboli chceme, aby business pojmy a logika klienta nebyly obsaženy ve službě DSpacu. Např. nechceme, aby se v kontraktu služby pro updatovani struktury DSpacu objevily pojmy typu „katedra" nebo „fakulta" (pojmy specifické pro Výměník), nebo aby se v implementaci služby vyskytl příkaz typu: „if komunita (pojem DSpacu) je katedra (pojem Výměníku) then něco".

Ze splnění těchto požadavků okamžitě vyplývá logická granularita. Neboli vznikne nová vrstva abstrakce (kontrakty) na které lze formulovat (komponovat) nové procesy bez jakékoliv znalosti implementace kontraktů (analogie např. interfaců v Javě).

4 Alternativy řešení a jejich srovnání ve světle technologických a metodologických požadavků.

4.1 Přímé použití stávajícího API DSpacu.

Toto nejvíce přímočaré řešení spočívá v tom, že klient přistupuje přímo k programovému API DSpacu.

Výhody:

- Toto řešení je nejméně časově náročné, protože kód řeší bezprostředně business logiku dané úlohy.

Nevýhody:

- Klientská aplikace se de facto stává součástí DSpacu v tom smyslu, že její kód musí běžet na stejné virtuální mašině Javy (dokonce ve stejném Classloaderu), jako DSpace. Z toho vyplývá, že klient je silně fyzicky a „Javovsky“ propojen s DSpacem, což odporuje požadavkům 3.1.1 a 3.1.2.
- V eventuálně nových verzích API DSpacu může dojít k nekompatibilitě se stávajícími klienty.
- Z faktu, že se klienti stávají součástí DSpacu vyplývá, že řešení nevyhovuje požadavku 3.2.2 (oddělení logiky klientů od DSpacu). Proč by měl DSpace např. rozumět významu sloupce LIC_SML_ZVEREJNIT_ROKY v pohledu ST_DSPACE_VSKP_VW KOSovské databáze?

4.2 Použití utilit poskytovaných DSpacem.

DSpace poskytuje sadu utilit (javovských programů), mj. programy pro importování struktury a importování itemů.

Výhody:

- Utility poskytují rozhraní (kontrakt), čímž oddělíme logiku klientů od logiky DSpacu (požadavek 3.2.2).
- Předpokládáme, že i v eventuálních nových verzích bude toto rozhraní akceptováno.

Nevýhody:

- Tvar rozhraní poskytovaných utilit nevyhovuje zcela naším požadavkům. Např. potřebujeme uvést globální identifikátor komunit, který bude spravován mimo DSpace (předpokládáme, že jedinečné místo spravující strukturu univerzity je Výměník).
- Ne vždy je rozhraní (kontrakt) definován jasně a přesně (např. pro importování struktury není definice kontraktu uvedena vůbec) – požadavek 3.2.1.
- Vstupy utilit jsou požadovány v souboru na přístupném file systému. Co když nemáme přístup na file systém DSpacu? – požadavek 3.1.1.
- Utility problém použití různých policies v různých kontextech (požadavek 3.1.4) neřeší.

4.3 Obecné SOA řešení implementované webovými službami

Řešení je postavené na definici služeb DSpacu, která vychází z principů Servisově orientovaného paradigmatu (SOA). Jednotlivé služby se implementují technologií webových služeb (Web Services).

Výhody:

Všechny požadavky uvedené v kapitole 3 jsou obsaženy v principech SOA paradigmatu. Následuje seznam některých principů (uvedeny zde, knižní podoba v knihovně VIC), které tyto požadavky pokrývají a popis, jakým webové služby tyto principy realizují. Vžité označení principů z angličtiny nepřekládáme, aby nevznikaly zmatky.

- **Service interoperability**: Neboli chceme, aby libovolné dva servisy spolu dokázaly „mluvit“. Tento princip je implicitně skryt ve všech našich požadavcích a explicitně je vyjádřen v požadavcích 3.1.1 a 3.1.2. Webové služby toto samozřejmě umožňují. Používají společně SOAP protokol a způsob

transportu zpráv (HTTP, JMS…) definují ve svém kontraktu (WSDL).

- **Standardized service contract**: Kontrakty webových služeb definujeme pomocí WSDL dokumentů (požadavek 3.2.1). Dále pak, WS-Policy (součást WSDL) umožňuje definovat jednotlivé policies (požadavek 3.1.4), kterými definujeme chování služby v různých kontextech.

- **Service Loose Coupling**: Neboli chceme, aby byl klient se servisou „co nejvolněji" svázán a to na různých úrovních. Na fyzické úrovni realizováno pomocí vzdáleného volání servisů pomocí platformově nezávislého, všeobecně podporovaného transportu (např. HTTP) – požadavky 3.1.1 a 3.1.2. Dále ve WSDL oddělujeme business část kontraktu (PortType) od definice způsobu přenášení zpráv mezi servisou a klientem (bindings) – požadavek 3.1.3. Pro jeden PortType můžeme definovat různé bindings. Na logické úrovni je tento princip vyjádřen požadavkem 3.2.2 (oddělení logik servisu a klienta).

- **Service Abstraction, Autonomy, Reusability and Composability:** Tyto obecné principy se všelijak prolínají a vyplívají jedny z druhých. Jsou součástí základních pojmů „SOA myšlení, vyjadřování, hodnocení". Např. čím přesnější kontrakt, tím zřetelnější míra autonomie. Čím větší autonomie, tím větší znovupoužitelnost a naopak. Čím větší znovupoužitelnost služeb, tím lépe se komponují a naopak. Komponovaná služba má větší míru abstrakce. Různé vrstvy abstrakce umožňují granulovatelnost pohledů na služby, a tedy usnadňují jejich znovupoužitelnost atd. Konkrétně v našem případě jsou tyto principy vyjádřeny požadavky v kapitole 3.2.

Nevýhody:

- Funkcionalitu DSpacu je nutné obalit webovými službami, což si vyžaduje určitý čas. Nicméně tato jednorázová vstupní investice se bohatě vrátí při eventuálním dalším rozvoji.

5 Popis stávajícího řešení

5.1 Identifikace servisů

Po rozboru uvedeném v kapitole 4 jsme se rozhodli implementovat obecné SOA řešení (kapitola 4.3). V DSpacu jsme identifikovali tyto servisy (služby):

- **GetStructure** – Servis vrací úplnou strukturu komunit DSpacu. Předpokládáme, že se nemusí nutně jednat o strom, ale o obecný graf. Proto je struktura definovaná množinou vrcholů a hran.

- **UpdateStructure** - Servis updatuje strukturu v DSpacu. Definujeme tyto operace pro updatovani:
    - **Vrcholy:**
        - Insert (vrchol)
        - Update (vrchol)
        - Delete (Id vrcholu)
        - CloneUpdate (vrchol)
    - **Hrany:**
        - Insert(hrana)

- Delete(hrana)

Obecný význam jednotlivých operací je zřejmý, až na operaci CloneUpdate. Tato operace replikuje stávající vrchol (včetně hran) a hodnoty atributů nového vrcholu přepíše hodnotami atributů vrcholu z argumentu operace.

Potřeba operace CloneUpdate vyplynula z knihovnických požadavků. Např. při přejmenovávání kateder je nutné uchovávat historii struktury. Bylo dohodnuto, že tato historie bude společně s aktuálním stavem v jedné struktuře. Jelikož nechceme replikovat celé podstromy přejmenovávaných uzlů, operace cloneUpdate replikuje pouze měněný vrchol a synové původního vrcholu se stávají zároveň syny nového vrcholu. Z toho vyplývá, že struktura po provedení této operace již nemusí být strom, ale obecný graf. Práci s obecným grafem bylo nutno v DSpacu doimplementovat.

Kontrakt operace Delete u vrcholu je obecný – prostě „nějak" vrchol smaž. V naší implementaci vrchol ponecháme ve struktuře (historie), pouze ho označíme, odkdy se stal neplatným. Totéž učiníme s původním vrcholem v operaci cloneUpdate. Toto je implementační detail, který v kontraktu není obsažen, takže tuto servisu mohou eventuálně poskytovat i jiné repository, s jinou implementací.

- **ImportStructure** - Servis importuje stromovou strukturu komunit DSpacu.
- **GetCollection** – Servis vyhledává handle DSpacové kolekce na základě jejího jména a externího identifikátoru příslušné komunity.
- **PostItem** – Servis importuje itemy do DSpacu. DSpace tento servis implementuje pomocí SWORD protokolu a HTTP jako transportní vrstvy (voláním servletu). Nejedná se tedy o WebService z čehož plynou některá omezení. V tuto chvíli je však toto řešení dostačující.

5.2 Implementace servisů

Důsledně od sebe oddělujeme tyto vrstvy:

- **Servisní vrstva:** Autentizace requestů, Parsování a případná validace requestů/responsů, jejich transformace na business objekty, delegování do business vrstvy, převedení chyb do tvaru v SOAP fault message.
- **Business vrstva:** Vlastní funkcionalita servisů, buď volaná ve stávajícím DSpacu nebo doimplementovaná.
- **Datová vrstva:** Oddělená použitím DAO patternu. Implementovaná pomocí iBATIS frameworku (mapování SQL na javovské struktury).

Implementace všech vrstev používá Spring framework (moduly Core a WS). Výsledný artefakt je webová aplikace (war file) deployovatelný na libovolném javovském webovém kontejneru (v našem případě Tomcat).

Všechny služby jsou implementovány synchronně přes HTTP transport.

5.3 Implementace klientů

Klienti webových služeb jsou implementovány pomocí Spring frameworku (moduly Core a WS). Databázové vrstvy používají framework iBATIS. Výsledné artefakty tvoří javovské aplikace v „jar file"

tvaru.

- Import struktury Výměníku do DSpacu: Klient načte stromovou strukturu Výměníku a importuje jí pomocí služby **ImportStructure**.

- Synchronizace Výměníku se strukturou DSpacu: Klient načte strukturu DSpacu voláním služby **GetStructure**. Potom načte strukturu z Výměníku a obě porovná. Ze zjištěných změn vygeneruje request, kterým volá službu **UpdateStructure**.

- Import itemů do DSpacu: Klient načte z KOSu importované itemy. Pro každý item vytvoří descriptor v METS formátu, ve kterém je popsaná struktura itemu (struktura obsažených souborů, jejich metadata (v Doublin Coru), pravidla jejich zvěřejňování atd.). Tento soubor společně se soubory tvořícími item zkopíruje do pomocného adresáře itemu. Obsah adresáře zkomprimuje do .zip souboru, který odešle SWORD protokolem do DSpacu – volání služby **PostItem**. Identifikátor příslušné kolekce, kam item patří, vyhledá pomocí služby **GetCollection**. Pokud import proběhne v pořádku, updatuje item v KOSu do stavu „importován“.

Vycházeli jsme z této open sourcové implementace SWORD klienta, kterou jsme upravili.

6 Co můžeme vylepšit?

- Služba **PostItem** je volána synchronně v jedné HTTP konekci a tento transport je natvrdo naimplementován jak ve službě samotné, tak v jejím klientovi. V Požadavku 3.1.3 ukazujeme, že v případě, kdy vyprší timeout konekce a item se přitom uloží korektně, klient se to nedoví a vznikne inkonzistentní stav. Vzhledem k možné velikosti itemů je tato situace reálná. Řešením je poskytovat službu **PostItem** asynchronně. Avšak z logiky služby v procesu importování itemu vyplývá její synchronnost – importuj item, vrať handle na vzniklý item v DSpacu, ulož handle do KOSu, updatuj item v KOSu na importovaný - proces tedy čeká, až služba vrátí handle itemu. Jde nám tedy pouze o to poskytnout asynchronně transportní vrstvu služby. Toho docílíme jednoduše např. použitím Correlation Id patternu – request message se uloží do request fronty a proces je suspendován do okamžiku, kdy se v response frontě objeví odpověď (identifikována atributem Correlation Id, jehož hodnota je rovna id request message).

- Z výše popsaného procesu importování itemu do DSpacu vyplývá, že by měl proběhnout v jedné distribuované transakci (uložení itemu v Dspacu a updatování KOSu). Toho lze dosáhnout např. použitím WS-transaction policy (nutno ovšem stávající Službu PostItem obalit WebServisou nebo proprietárně vyřešit jinak). V současném stavu, pokud z nějakého důvodu proběhne commit v DB DSpacu a nikoliv v DB KOSu, vznikne inkonzistentní stav.

- Službu **UpdateStructure** můžeme volat asynchronně. Navíc může být více komponent, které bude změna struktury zajímat. V takovém případě můžeme použít Publish/Subscribe pattern – Výměník posílá (publikuje) změny ve formě JMS message do nějaké fronty a už se o ně nestará. Komponenty, které to zajímá (subscribers – např. DSpace) tyto zprávy čtou a nějak zpracovávají. Když je zpracují všichni zaregistrovaní subscribeři, zpráva se smaže.

Efektivní možnosti implementace výše uvedených zlepšení poskytuje ESB (Enterprise Service Bus – např. open sourcový JBoss ESB). Jeho použití předpokládá instalaci Java aplikačního serveru (např. open sourcový JBoss AS).

Appendix A

- SWORD (Simple Web-service Offering Repository Deposit)
    - Pro ukládání depositů do repository
        - Podpora komprimování strukturovaných depositů
        - Otevřeno pro různé způsoby strukturování depositů – např. METS
        - Podpora mediace
    - Pro zjišťování obsahu repository
        - Vychází z AtomPub protokolu a Atom formátu
            - Spravování blogů
            - REST (Representational State Transfer)
                - Orientace na podstatná jména (věci)
                - Každá věc jednoznačně identifikována pomocí URI
                - Používají se pouze standardní http metody (GET, POST, PUT, DELETE)
                - Věci propojeny (pomocí hyperlinků)
                - Věci mají různé reprezentace (Html, Atom)
                - Komunikace je bezestavová
                - REST vs. WebServices
- Potřebujeme přidělovat práva přístupu k jednotlivým souborům itemu DSpacu. Proto jsme definovali formát přístupových práv, který používáme v administrativní sekci formátu METS (element *admSec*). Tento formát obsahuje odkazy na soubory, role a jejich práva (v DSpacu nejsou role, takže jsme ztotožnili role se skupinami uživatelů - EPersonGroups). Každé právo obsahuje typ akce (read/write) a interval platnosti. Zpracování těchto práv bylo nutné doimplementovat v DSpacu, takže jsme přepsali METS ingestor plug-in. Následuje ukázka definice práv v METS:

```
<amdSec ID="sword-mets-amd-1">
    <rightsMD GROUPID="sword-mets-amd-1_group-1"
        ID="sword-mets-rights-1">
        <mdWrap MIMETYPE="text/xml" OTHERMDTYPE="EPDCX"
            MDTYPE="OTHER" LABEL="SWAP Metadata">
            <xmlData>
                <ns4:fileRightsList>
                    <ns4:fileRightsType>
                        <ns4:fileId>sword-mets-file-2</ns4:fileId>
                        <ns4:resourcePolicy>
                            <ns4:action>READ</ns4:action>
                                    <ns4:role>DSpaceUser</ns4:role>
                            <ns4:startDate>
                                2011-02-19T00:00:00.000-08:00
```

```
                    </ns4:startDate>
                    <ns4:endDate>
                        2999-12-31T00:00:00.000-08:00
                    </ns4:endDate>
                </ns4:resourcePolicy>
                <ns4:resourcePolicy>
                        <ns4:action>READ</ns4:action>
                    <ns4:role>Anonymous</ns4:role>
                    <ns4:startDate>
                        2011-02-19T00:00:00.000-08:00
                        </ns4:startDate>
                    <ns4:endDate>
                        2999-12-31T00:00:00.000-08:00
                    </ns4:endDate>
                </ns4:resourcePolicy>
            </ns4:fileRightsType>
        </ns4:fileRightsList>
    </xmlData>
</mdWrap>
</rightsMD>
</amdSec>
```